$AF/2173

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application No.: 09/726,779	§	
Filed: November 29, 2000	§	Examiner: Pillai, Namitha
Inventor(s):	§	Group/Art Unit: 2173
Chris Cifra, Kevin Schultz, Jeff Kellam, Jeff Correll, Nicolas Vazquez, And Christophe Caltagirone	§	Atty. Dkt. No: 5150-43700
Title: AUTOMATIC GENERATION OF PROGRAMS WITH GUI CONTROLS FOR INTERACTIVELY SETTING OR VIEWING VALUES	§	

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to Commissioner for Patents, Alexandria, VA 22313-1450, on the date indicated below.

Jeffrey C. Hood

Signature 5/5/2006 Date

APPEAL BRIEF

Box: Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir/Madam:

Further to the Notice of Appeal filed March 6, 2006, Appellant presents this Appeal Brief. Appellant respectfully requests that this appeal be considered by the Board of Patent Appeals and Interferences.

I. REAL PARTY IN INTEREST

The subject application is owned by National Instruments Corporation, a corporation organized and existing under and by virtue of the laws of the State of Delaware, and having its principal place of business at 11500 N. MoPac Expressway, Bldg. B, Austin, Texas 78759-3504.

II. RELATED APPEALS AND INTERFERENCES

No related appeals or interferences are known which would directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

III. STATUS OF CLAIMS

Claims 1 – 45 were originally filed in the application. In an amendment filed November 13, 2003, claims 2, 15, 28, and 41 were cancelled for not further limiting the claimed invention. In an amendment filed April 21, 2004, claims 1, 4-6, 8, 10, 12-14, 17-19, 21, 23, 25-27, 30-32, 34, 36, 38-40, 43, and 44 were amended, and new claims 46 and 47 were added. In an amendment filed March 24, 2005, claims 1, 12, 14, 27, and 38-40 were amended, and new claims 48-55 were added. In an amendment filed September 30, 2005, claims 1, 3-14, 16-18, 20-21, 24-27, 29-35, 37-40, 42, 44, 46-52, and 55 were amended, and new claims 56-59 were added. Thus, claims 1, 3-14, 16-27, 29-40, and 42-59 are currently pending in the application

Claims 1, 3-5, 7-14, 16-18, 20-27, 29-31, 33-40, and 42-59 stand rejected under 35 U.S.C. § 102(e), and claims 6, 19, and 32 stand rejected under 35 U.S.C. § 103(a), and are the subject of this appeal. A copy of claims 1-59, incorporating entered amendments, as on appeal, is included in the Claims Appendix hereto.

IV. STATUS OF AMENDMENTS

No amendments to the claims have been filed subsequent to the rejection in the Office Action of January 30, 2006. The Claims Appendix hereto reflects the current state of the claims.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

The present application relates to field of automatic computer program generation, e.g., program prototyping, and more particularly, to the automatic generation of a computer program from program information, e.g., from a specified prototype.

Independent claim 1 recites a method for generating a computer program from a prototype. *See, e.g., Specification p. 2, lines 13–15, p. 7, lines 13 – 17.* User input is received to a prototyping application, where the user input specifies a prototype, e.g., a recipe, solution, or script, including a series of functional operations, and where the functional operations include a first functional operation with (one or more) associated parameters. *See, e.g., Specification p. 14, line 28 – p.15, line 20, p. 22, lines 25 – 30; p. 24, line 26 – p. 25, line 30; Figure 5 (300-306), Figures 6-12.* In response to the specified prototype, e.g., based on the specified prototype, a program is automatically generated that implements the prototype, i.e., that implements the specified solution. *See, e.g., Specification p. 26, lines 1 – 20, p. 26, line 29 – p. 27, line 4; Figure 5 (308), Figures 14-19, Figure 21.* The program is operable to execute independently of the prototyping application. In other words, the prototyping application is not required to execute the automatically generated program. *See, e.g., Specification p. 8, lines 9 – 12.*

Automatically generating the program also includes automatically generating a graphical user interface for the program, including automatically creating one or more graphical user interface elements associated with the (one or more) parameters of the first functional operation. *See, e.g., Specification p. 2, lines 11 – 13, p. 7, lines 4 – 5 and lines 13 – 17, p. 8, line 21 – p. 9, line 2, p. 9, lines 11 – 19, p. 9, line 27 – p. 10, line 2, p. 22, lines 19 – 22.* During execution of the program, at least one of the graphical user interface elements is displayed and is operable to receive user input independently of the prototyping application. *See, e.g., Specification p. 8, lines 9 – 12, p. 25, lines 5 – 8, p. 26, lines 24 – 26, p. 38, line 12 – 24, p. 40, line 29 – p. 41, line 6, p. 41, lines 14 - 26; Figure 22.*

Independent claim 14 is an analogue system claim to method claim 1, and includes substantially the same limitations as claim 1, summarized above. More particularly, the system includes a computer program comprising a prototyping environment that is executable to perform the method of claim 1. *See, e.g., the citations above regarding claim 1, as well as, e.g., Specification p. 14, line 23 – p. 15, line 29, p. 16, line 4 – p. 17, line 12, p. 17, line 16 – p. 22, line 2; Figures 1, 2A, 2B, and 3.*

Independent claim 27 is an analogue memory medium claim to method claim 1, and includes substantially the same limitations as claim 1, summarized above. In other words, the memory medium stores program instructions executable to perform the method of claim 1. *See, e.g., the citations above regarding claim 1, as well as, e.g., Specification p. 20, lines 6 – 21, Figure 3.*

Independent claim 40 is directed to a method for automatically generating a computer program based on received program information. Program information is received to a first application, e.g., a prototyping application, text executive program, etc., where the program information specifies the functionality of the computer program. *See, e.g., Specification p. 7, lines 2 – 12, p. 22, lines 9 – 13, p. 23, lines 1 - 3; Figure 4 (900).* A computer program implementing the specified functionality is automatically generated in response to the program information. *See, e.g., Specification p. 7, lines 2 – 5, p. 22, lines 14 – 18; Figure 4 (902).* The automatically generated program is executable independently of the first application. In other words, the first application is not required to execute the automatically generated computer program. *See, e.g., Specification p. 8, lines 9 – 12.*

Automatically generating the computer program also includes automatically generating a graphical user interface for the program, including automatically creating (one or more) graphical user interface elements for providing input to and/or viewing output from the program. *See, e.g., Specification p. 2, lines 11 – 13, p. 7, lines 4 – 5 and lines 13 – 17, p. 8, line 21 – p. 9, line 2, p. 9, lines 11 – 19, p. 9, line 27 – p. 10, line 2, p. 22, lines 19 – 22; Figure 9 (904).* During execution of the program, the (one or more) graphical user interface elements are displayed, and at least one of the graphical user

interface elements is operable to receive user input independently of the first application. *See, e.g., Specification p. 8, lines 9 – 12, p. 25, lines 5 – 8, p. 26, lines 24 – 26, p. 38, line 12 – 24, p. 40, line 29 – p. 41, line 6, p. 41, lines 14 - 26; Figure 22.*

Independent claim 48 is directed to a method for generating a computer program, specifically, a graphical program. User input specifying a prototype, e.g., a recipe, solution, or script, is received, where the prototype includes a series of functional operations, at least one of which has an associated one or more parameters. *See, e.g., Specification p. 14, line 28 – p.15, line 20, p. 22, lines 25 – 30; p. 24, line 26 – p. 25, line 30; Figure 5 (300-306), Figures 6-12.* In response to the user input specifying the prototype, a graphical program is automatically generated, e.g., that implements the prototype, i.e., that is interpretable or compilable, and executable to perform the functional operations. *See, e.g., Specification p. 26, lines 1 – 20, p. 26, line 29 – p. 27, line 4; Figure 5 (308), Figures 14-19, Figure 21.* Automatically generating the graphical program includes automatically generating a plurality of interconnected nodes that visually indicate functionality of the graphical program, where the plurality of interconnected nodes are operable to perform the series of functional operations. *See, e.g., Specification p. 26, lines 10 - 20, p. 36, line 24 – p. 37, line 11; Figures 20, 21*

Automatically generating the graphical program also includes automatically generating a graphical user interface for the graphical program, wherein the graphical user interface for the graphical program includes at least one graphical user interface element that is associated with at least one of the one or more parameters. *See, e.g., Specification p. 2, lines 11 – 13, p. 7, lines 4 – 5 and lines 13 – 17, p. 8, line 21 – p. 9, line 2, p. 9, lines 11 – 19, p. 9, line 27 – p. 10, line 2, p. 22, lines 19 – 22; p. 40, line 29 – p. 41, line 6; Figure 22.*

Independent claim 50 is directed to a method for generating a computer program based on a user-specified prototype. User input is received, e.g., to a prototyping application, where the user input specifies a prototype, e.g., a recipe, solution, or script, including a series of functional operations, and where at least one of the operations has an associated one or more parameters. In other words, the user specifies a series or

sequence of functions to be performed, where at least one of the functions is associated with, includes, or uses one or more parameters. *See, e.g., Specification p. 14, line 28 – p.15, line 20, p. 22, lines 25 – 30; p. 24, line 26 – p. 25, line 30; Figure 5 (300-306), Figures 6-12.*

In response to the user input specifying the prototype, a graphical program is automatically generated, i.e., that implements the specified solution, including automatically generating a plurality of interconnected nodes that visually indicate functionality of the graphical program. *See, e.g., Specification p. 26, lines 1 – 20, p. 26, line 29 – p. 27, line 4, p. 36, line 24 – p. 37, line 11; Figure 5 (308), Figures 14-19, Figures 20, 21.*

Automatically generating the graphical program also includes automatically generating a graphical user interface for the graphical program. At least one of the parameters is then associated with an element of the graphical user interface. *See, e.g., Specification p. 2, lines 11 – 13, p. 7, lines 4 – 5 and lines 13 – 17, p. 8, lines 4 – 8, p. 8, line 21 – p. 9, line 2, p. 9, lines 11 – 19, p. 9, line 27 – p. 10, line 2, p. 22, lines 19 – 22; p. 40, line 29 – p. 41, line 6; Figure 22.*

Claim 53 is directed to a method for generating a computer program based on a user-specified prototype. A prototyping environment user interface is displayed on a display of a computer system, where the prototyping environment user interface is usable to create a prototype, e.g., a recipe, solution, or script. *See, e.g., Specification p. 7, lines 17 – 25, p. 8, line 2 – 8.* User input specifying the prototype is received, where the prototype includes a series of functional operations, at least one which has an associated one or more parameters. In other words, the user specifies a series or sequence of functions to be performed, where at least one of the functions is associated with, includes, or uses one or more parameters. *See, e.g., Specification p. 14, line 28 – p.15, line 20, p. 22, lines 25 – 30; p. 24, line 26 – p. 25, line 30; Figure 5 (300-306), Figures 6-12.*

A program that implements the prototype is automatically generated in response to the specified prototype, including a graphical user interface for the program. *See, e.g., Specification p. 7, lines 2 – 5, p. 22, lines 14 – 18; Figure 4 (902).* The graphical user

interface of the program includes at least one graphical user interface element that is associated with at least one of the associated one or more parameters, and that is operable to receive information to the program and/or to output information from the program during execution of the program. *See, e.g., Specification p. 2, lines 11 – 13, p. 7, lines 4 – 5 and lines 13 – 17, p. 8, line 21 – p. 9, line 2, p. 9, lines 11 – 19, p. 9, line 27 – p. 10, line 2, p. 22, lines 19 – 22; p. 40, line 29 – p. 41, line 6; Figure 22.* The graphical user interface of the program is independent of the prototyping environment user interface. *See, e.g., Specification p. 8, lines 9 – 12.*

Independent claim 55 is directed to a method for generating a computer program based on user input. User input specifying a series of functional operations is received to a development environment, where at least one of the operations has an associated one or more parameters. *See, e.g., Specification p. 14, line 28 – p.15, line 20, p. 22, lines 25 – 30; p. 24, line 26 – p. 25, line 30; Figure 5 (300-306), Figures 6-12.* A program that implements the series of functional operations is automatically generated in response to the user input, i.e., where the program is interpretable or compilable, and executable to perform the specified functional operations. Moreover, program execution of the program is independent of execution of the development environment. In other words, the development environment is not required to execute the automatically generated program. *See, e.g., Specification p. 7, lines 2 – 5, p. 8, lines 9 – 12, p. 22, lines 14 – 18; Figure 4 (902).*

Automatically generating the program also includes automatically generating a graphical user interface for the program, where the graphical user interface includes at least one graphical user interface element that is associated with at least one of the associated parameters, and that is operable to receive information to the program and/or to output information from the program during execution of the program *See, e.g., Specification p. 2, lines 11 – 13, p. 7, lines 4 – 5 and lines 13 – 17, p. 8, lines 4 – 8, p. 8, line 21 – p. 9, line 2, p. 9, lines 11 – 19, p. 9, line 27 – p. 10, line 2, p. 22, lines 19 – 22; p. 40, line 29 – p. 41, line 6; Figure 22.*

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Claims 1, 3-5, 7-14, 16-18, 20-27, 29-31, 33-40, and 42-59 stand rejected under 35 U.S.C. § 102(e) as being unpatentable over U.S. Patent No. 6,298,474 B1 to Blowers et al.

Claims 6, 19, and 32 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent No. 6,298,474 B1 to Blowers et al.

VII. ARGUMENT

Section 102(e) Rejection

Claims 1, 3-5, 7-14, 16-18, 20-27, 29-31, 33-40, and 42-59 stand rejected under 35 U.S.C. § 102(e) as being unpatentable over U.S. Patent No. 6,298,474 B1 to Blowers et al (henceforth, "Blowers"). Appellant respectfully traverses this rejection for the following reasons. Different groups of claims are addressed under their respective subheadings.

Claims 1, 11, 14, 24, 27, 37, 43, 47, 53, 54, 55, 57, 58, 59

Claims 1, 11, 14, 24, 27, 37, 43, 47, 53, 54, 55, 57, 58, and 59 are separately patentable because the cited reference does not teach or suggest the limitations recited in these claims.

Blowers teaches a design engine or task sequencer engine 46 that is used to configure a desired sequence of functional tasks. A user creates the desired sequence by selecting graphical representations or icons from the tool boxes of FIG. 5. As the user is creating the sequence of functional tasks, the user can configure parameters of the functional tasks as illustrated in FIGS. 7 and 8 with respect to the blob and alignment vision tools, respectively. Once the desired sequence has been created, it can be stored or saved in a condensed method within an inspection sequence file 52 which is useable by the engine 46. The engine 46 takes the condensed stored sequence from the file 52 and executes it through the runtime screen of FIG. 9. (See Col. 8, line 61 – Col. 9, line 25)

Blowers does not teach the combination of elements recited in claim 1. In particular, Blowers does not teach receiving user input to a prototyping application to specify a prototype and automatically generating a program to implement the prototype, *wherein the automatically generated program is operable to execute independently of the prototyping application.* As described above, and previously, Blowers instead teaches that an engine 46 is used to configure a desired sequence of functional tasks, and the desired sequence is then executed by the engine 46. In the Response to Arguments, the Examiner incorrectly asserts that "the prototyping application is independent from the generated program", stating that the prototyping application "is the machine vision system which is separate from the components used for developing the control structure

that is then used in a separate machine vision system". Appellant respectfully submits that the components being separate from the prototyping application is not the same as the generated program being operable to execute independently from the prototyping application. Blowers is quite clear that engine 46 is used to both develop the program and to execute the program.

Blowers also does not teach the recited elements of, "wherein said automatically generating the graphical user interface comprises automatically creating one or more graphical user interface elements associated with the one or more parameters of the first functional operation, wherein during execution of the program, at least one of the one or more graphical user interface elements is displayed and is operable to receive user input *independently of the prototyping application*".

The Examiner cites Blower's interactively generated tree structure as being an automatically generated graphical user interface for an automatically generated program. However, as Blowers makes clear, the tree structure is *not* automatically generated, but rather is created manually via the user dragging and dropping icons representing various functions into the structure. Moreover, as may be seen in Figures 7-9, the tree structure is not independent of the development engine interface of Blowers, and so Blowers cannot teach this feature. In other words, in Blowers (per the Abstract and col. 3, lines 25-35), the user-selected first control program (pre-existing machine vision step or module) is linked to the user-selected hardware operating parameters to form the application. No actual program code is automatically generated that is independently executable, but rather a sequence file is generated based on the user-created tree structure, where the sequence file is usable *by the engine* to perform the specified sequence.

Thus, Blowers fails to teach all the limitations of claim 1. Thus, Appellant respectfully submits that the teachings of Blowers clearly do not meet the standard for anticipation which requires that the identical invention must be shown in as complete detail as is contained in the claims. *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989). Anticipation requires the presence in a single prior art reference disclosure of each and every element of the claimed invention, arranged as in the claim. *Lindemann Maschinenfabrik GmbH v. American Hoist & Derrick Co.*, 730 F.2d 1452, 1457, 221 USPQ 481, 485 (Fed. Cir. 1984). Appellants'

invention as recited in claim 1 is clearly not anticipated by Blowers, and so claim 1 is allowable.

Claims 3, 16, 29, and 42

In addition to the distinctions noted above in regard to claim 1, Blowers fails to teach "wherein said automatically generating the program comprises automatically generating source code for the program without direct user input specifying the source code", as recited in claim 3.

Blowers does not teach automatically generating *source code for a program.* More particularly, Blowers does not teach automatically generating source code for a program that is automatically generated in response to a prototype specified by user input. In fact, Blowers nowhere mentions or even hints at automatically generating source code based on a specified prototype.

In asserting that Blowers teaches this feature, the Examiner cites col. 2, lines 45-55, which reads:

> An object of the present invention is to provide a method and system for interactively developing application software for use in a machine vision system and computer-readable storage medium having a program for executing the method wherein the user teaches an imaging programming task without writing any code. Consequently, the user need not be a programmer. The user does not write a single line of code, but rather sets variables that the machine vision tools require interactively.

Appellant notes that the user not having to write code is not the same as automatically generating source code in response to a specified prototype.

In the Response to Arguments, the Examiner cites col. 4, lines 15-19, and asserts that the cited text not only teaches automatically generating source code, but also how input specified by the user is used to produce this code. Appellant respectfully disagrees. The cited passage reads:

> The idea behind interactively generating a computer program is to ensure that the programmer cannot make syntax errors in producing his code. Hence, the environment guides the programmer via menus and appropriate interface structure to produce code that is correct in syntax.

Appellant notes that Blowers actually teaches interactive development of a tree structure that sequences pre-existent code modules, e.g., COM modules, vision steps, etc., and nowhere describes or illustrates automatically generated *source code*. For example, subsequent to the above text, Blowers states:

> The method and system of the present application is different from the examples mentioned above in that it takes a standard tree view interface and modifies it to allow the integration of another standard, namely COM controls which can be linked in.
>
> Hence, a first unique feature of this invention is that any third party tools in machine vision or automation or otherwise that are written as COM controls can be seamlessly linked into this interactive environment.
>
> COM controls are becoming widely accepted and many are being produced by many companies. These are able to be linked together and used without writing code that can produce syntax errors. A control sequence is set up triggered on events, timers and variable settings. This interactive task sequencer environment reads the properties of the COM controls and allows these to be set or linked to other properties of other controls. The operator may select which method(s) to run on the control and events fired by controls are used to sequence the logic. Hence, the present invention sets how any third party COM control behaves by viewing the current standard interfaces. (col.4:31-52)

Similarly, in col. 8:28-40, Blowers states:

> Referring now to FIG. 3, there is illustrated in block diagram form various software and hardware components for interactively developing a graphical, control-flow structure and associated application software for use in the machine vision system 20 of FIG. 2 using the computer system of FIG. 1 without the need for a user to write any code. *The method and system of the present invention "marry" the ActiveX-COM standard with the commonly used "tree view" method of navigation and hierarchy.* The system allows for adding new machine vision functions. The design includes a method of controlling the flow of sequences based on conditional branches and forced routing changes. *(emphasis added)*

Appellant further notes that the Blower's engine produces a "condensed stored sequence" based on the interactively developed tree structure, and stores the sequence in an "inspection sequence file", which is usable by the engine to execute the various functions, e.g., where the stored sequence specifies the call sequence. More specifically,

as noted above, in Blowers (per the Abstract and col. 3, lines 25-35), the user-selected first control program (pre-existing machine vision step or module) is linked to the user-selected hardware operating parameters to form the application, and that no actual program code is automatically generated, but rather a sequence file is generated based on the user-created tree structure, where the sequence file is usable *by the engine* to perform the specified sequence.

Nowhere does Blowers mention automatically generating *source code*. Rather, Blowers provides a structure whereby prewritten code modules, e.g., COM modules, may be linked together and called in a specified sequence, in accordance with an interactively developed tree structure. Appellant respectfully submits that in Blower's system, no source code is automatically generated, particularly in response to a specified prototype.

Thus, for at least these reasons, Appellant submits that Blower's fails to teach or suggest all the limitations of claim 3, and so claim 3 is allowable.

Claims 4, 5, 12, 17, 25, 30, 31, 38

In addition to the distinctions noted above in regard to claim 1, Blowers fails to teach "wherein the first functional operation has an associated input parameter, wherein the one or more automatically created graphical user interface elements include a graphical user interface element for interactively receiving user input specifying a value for the input parameter", as recited in claim 4.

Blowers does not teach automatically creating graphical user interface elements at all, and more particularly does not teach automatically creating graphical user interface elements for interactively receiving user input specifying a value for the input parameter.

In asserting the Blowers teaches this features, the Examiner cites col. 3, lines 28-34, which reads:

> Then, the method includes receiving commands from a user to select desired hardware operating parameters corresponding to desired hardware and a machine vision graphical representation and its associated first control program corresponding to a desired machine vision task.

Appellant submits that the cited text nowhere discloses automatically creating graphical user interface elements for interactively receiving user input specifying a value

for the input parameter, where the automatically created graphical user interface elements are part of the automatically generated program. Rather, the cited text discloses manual selection by the user of hardware operating parameters (that correspond to desired hardware), and a node or icon (for the tree structure) corresponding to a (pre-existent) control program. For example, Appellant notes that the text immediately preceding the cited text reads:

> The method includes the step of providing a first set of control programs representing possible machine vision tasks. The first set of control programs define a first set of standard controls. The method also includes the step of providing hardware operating parameters corresponding to possible hardware. The hardware operating parameters define a second set of standard controls. The method further includes the step of displaying graphical representations of possible hardware and possible machine vision tasks.

As may be seen, all the control programs (representing the possible machine vision tasks) are already provided, and the user manually selects these programs (via their respective icons) for inclusion in the tree structure. Similarly, the hardware operating parameters (corresponding to possible hardware) are also already provided, and define standard controls, where the user selects those hardware parameters/controls desired for use in setting the values of the parameters. Nowhere are graphical user interface elements generated automatically as claimed, particularly that are operable independently of the (development) application.

The Examiner also cites Figure 7 and col. 9, lines 7-9. However, the cited text simply states that Blowers' tasks have parameters that are configurable. Appellant notes that the blob properties dialog shown in Figure 7 is part of the Blowers development environment, i.e., the development engine 46. This dialog box is *not* automatically created from the specified prototype. Moreover, Appellant notes that Figure 7 clearly shows that the configuration of the parameters is performed *within the framework of the engine*, e.g., via the *design engine's interface*, and is specifically *not* performed via an automatically generated graphical user interface that is part of an automatically generated program, that is automatically generated in response to a specified prototype, and that operates independently of the application. In fact, the further cited col. 8, lines 61-67, reads:

> A design engine or task sequencer engine 46 is used to configure and test the flow and design of the application software as illustrated by an exemplary task sequencer list of FIG. 6. Graphical representations or icons are selected from the tool boxes of FIG. 5 which correspond to desired functional tasks and are linked into the tree structure of FIG. 6 by a task sequencer interface 50 in the desired locations.

Thus, the parameter configuration is performed via the design or task sequencer engine 46, not via automatically generated graphical program elements.

Thus, for at least these reasons, Appellant respectfully submits that the cited text (and Blowers in general) fails to teach or suggest the features and limitations of claim 4, and so claim 4 is allowable. Similar reasoning applies to the features and limitations of claim 5.

Claims 7, 20, 33

In addition to the distinctions noted above in regard to claim 1, Blowers fails to teach "wherein the automatically generated program comprises a text-based program; wherein automatically generating the program comprises automatically generating source code for the program in a text-based programming language", as recited in claim 7.

Appellant submits that the cited text (col. 1, lines 33-36) describes traditional program development, e.g., in C++, Delphi, or Visual Basic, and fails to disclose automatically generating source code for the program in a text-based programming language, where the automatic generation is performed based on a specified prototype, as claimed. Nor does Blowers in general teach or suggest such automatic generation of text-based source code in a text-based language. Appellant notes that the cited Figure 6 of Blowers illustrates pre-existing computer files or modules included in the tree structure, and that are callable per the (user-interactively created) tree structure, not automatically created text-based source code.

Thus, for at least this reasons, Blowers fails to teach all the features and limitations of claim 7, and so claim 7 is allowable.

Claims 8, 21, 34, 44, 46

In addition to the distinctions noted above in regard to claim 1, Blowers fails to teach "wherein the automatically generated program comprises a graphical program; wherein automatically generating the program comprises automatically generating graphical source code for the graphical program, wherein the graphical source code includes a plurality of interconnected nodes that visually indicate functionality of the graphical program", as recited in claim 8.

In asserting that Blowers teaches this feature, the Examiner cites col. 3, lines 40-45, which reads:

> Then, the method includes linking the selected first control program with the desired hardware operating parameters to form the application software in response to the commands without the user writing any of the application software.

Nowhere does the cited text (or Blowers in general) teach or suggest automatically generating a graphical program, including graphical source code. Additionally, Blowers is unclear as to what exactly the "application software" formed by linking the user-selected control program with the user-selected hardware operating parameters is. Moreover, nowhere does Blowers disclose automatically generating graphical source code of a graphical program in response to a specified prototype, as claimed.

The Examiner also cites Figure 6 as teaching this feature. However, Figure 6 illustrates an exemplary task sequencer list configured by the design engine or task sequencer engine 46. (*See Col. 8, lines 61-67*) As discussed above, the sequence of tasks is created in response to user input selecting graphical representations or icons from the tool boxes of FIG. 5. Figure 6 does not illustrate a program that is ***automatically generated*** in response to a prototype specified by user input, as recited in claims 1 and 8. In other words, Blowers' tree structure is not generated automatically, but rather, is created manually by the user dragging and dropping module icons into the structure, or issuing commands to that effect. (*See, e.g., col. 3, lines 25-35.*)

Appellant respectfully submits that the Examiner has improperly speculated as to the nature of Blowers' application software, and has mischaracterized Blowers' tree structure, and respectfully submits that Blowers fails to teach or suggest all the features and limitations of claim 8, and so claim 8 is allowable.

Claims 9, 22, 35

In addition to the distinctions noted above in regard to claim 1, Blowers fails to teach "wherein the prototyping application interfaces with a programming environment application in order to perform said generating the program", as recited in claim 9.

In asserting that Blowers teaches this feature, the Examiner cites col. 3, lines 40-44, which reads:

> Then, the method includes linking the selected first control program with the desired hardware operating parameters to form the application software in response to the commands without the user writing any of the application software.

As may be seen, the cited text makes no mention of a prototyping application interfacing with a programming environment application to automatically generate a program (in response to a specified prototype), and so does not teach or suggest these features of claim 9, and so claim 9 is allowable, for at least this reason.

Claims 10, 23, 36

In addition to the distinctions noted above in regard to claim 1, Blowers fails to teach "wherein the first functional operation has a first parameter that has an associated data type, the method further comprising: determining the data type of the first parameter; wherein automatically creating the one or more graphical user interface elements comprises automatically creating a first graphical user interface element associated with the first parameter, wherein the first graphical user interface element is operable to receive user input according to the data type of the first parameter.", as recited in claim 10.

In asserting that Blowers teaches this feature, the Examiner cites Figure 6. However, Figure 6 does not disclose automatically creating a first graphical user interface element associated with the first parameter, wherein the first graphical user interface element is operable to receive user input according to the data type of the first parameter. Appellant notes that the cited "Blob" and "Acquire" icons selected by the user to represent control programs, e.g., code modules, and are not automatically generated

graphical user elements of an automatically generated graphical user interface. For example, the "Blob" icon represents a blob analysis to be performed on an image, not a blob data type. Moreover, as noted above, the tree structure of Blowers is not generated automatically, but rather via the user manually dragging and dropping icons into the structure. Finally, nowhere does Blowers disclose automatically generating graphical user interface elements that operate independently of the development application, e.g., the design engine.

Thus, Appellant respectfully submits that Blowers fails to teach this feature of claim 10, and so claim 10 is allowable.

Claims 13, 18, 26, 39

In addition to the distinctions noted above in regard to claim 1 (and 11), Blowers fails to teach "wherein the one or more automatically created graphical user interface elements include a first graphical user interface element for viewing an output value for an output parameter of the first functional operation, wherein the output value is determined by the image processing algorithm", as recited in claim 13.

In asserting that Blowers teaches this feature, the Examiner cites col. 3, lines 45-55, which reads:

> Preferably, the application processing engines include a results engine for generating and storing records within a database based on the results.
>
> The desired hardware operating parameters may correspond to a desired image source such as a video camera. The results engine stores images from the video camera within the database based on the results.
>
> Preferably, the method further comprises the step of graphically displaying the results within a rolling results window.

The cited text describes displaying results from the application in a rolling results window of the design engine interface. Nowhere does the cited text disclose automatically created a graphical user interface element for viewing output from a function, where the graphical user element is part of an automatically generated graphical user interface that operates independently from the prototyping application, as claimed.

Thus, Blowers fails to provide this feature of claim 13, and so claim 13 is allowable.

Claims 40

Claim 40 is separately patentable because the cited reference does not teach or suggest the limitations recited in this claim.

As noted above with respect to claim 1, Blowers teaches a design engine or task sequencer engine 46 that is used to configure a desired sequence of functional tasks. A user creates the desired sequence by selecting graphical representations or icons from the tool boxes of FIG. 5. As the user is creating the sequence of functional tasks, the user can configure parameters of the functional tasks as illustrated in FIGS. 7 and 8 with respect to the blob and alignment vision tools, respectively. Once the desired sequence has been created, it can be stored or saved in a condensed method within an inspection sequence file 52 that is useable by the engine 46. The engine 46 takes the condensed stored sequence from the file 52 and executes it through the runtime screen of FIG. 9. (See Col. 8, line 61 – Col. 9, line 25)

Blowers does not teach the combination of elements recited in claim 40. In particular, Blowers does not teach receiving program information to a first application to specify program functionality and automatically generating a program to implement the specified functionality, *wherein the program is operable to execute independently of the first application.* As described above, and previously, Blowers instead teaches that an engine 46 is used to configure a desired sequence of functional tasks, and the desired sequence is then performed or executed by the engine 46. The Examiner incorrectly asserts that "the program is operable to execute independently of the prototyping application...there being a distinctiveness between the program that carries out the task through the structure created in the hardware configuration and the machine vision system", citing col. 3, lines 35-45, and col. 4, lines 1-15. Appellant respectfully submits that the program that carries out the machine vision task through the structure *is* the machine vision system, and notes that Blowers is quite clear that engine 46 is used to both develop the program and to execute the program. In other words, per Blowers, the

software application *requires* the engine to execute, and so Blowers does not teach this claimed feature.

Blowers also does not teach the recited elements of, "wherein said automatically generating the graphical user interface comprises automatically creating one or more graphical user interface elements for providing input to and/or viewing output from the program, wherein during execution of the program, the one or more graphical user interface elements are displayed and at least one of the one or more graphical user interface elements is operable to receive user input *independently of the first application*", since Blowers requires the development engine to present the graphical user interface elements. The Examiner argues that "the user is allowed to input during execution without interference from the prototyping application thereby showing independence". However, being able to "input during execution without interference from the prototyping application" is *not* the same as being independent, since, as argued above, Blower's development engine is clearly required to operate or present graphical user elements to receive user input during execution. Thus, Blowers fails to provide this claimed feature.

Thus, for at least the reasons provided above, Appellant respectfully submits that Blowers fails to teach or suggest all the features and limitations of claim 40, and so claim 40 is allowable.

Claim 45

In addition to the distinctions noted above in regard to claim 40, Blowers fails to teach "wherein the received program information specifies one of: a prototype; a test executive sequence; and a state diagram", as recited in claim 45.

In asserting that Blowers teaches this feature, the Examiner cites Figure 6, particularly pointing out the "if-then-else" statement as representing a test executive sequence and a state diagram. Appellant respectfully submits that "if-then-else" statements are well known standard computer program constructs, and are neither a test executive sequence nor a state diagram. Thus, Blowers fails to teach this claimed feature, and so claim 45 is allowable.

Claim 48, 50, 51

Claims 48, 50, and 51 are separately patentable because the cited reference does not teach or suggest the limitations recited in these claims.

As submitted above, Blowers teaches a design engine or task sequencer engine 46 that is used to configure a desired sequence of functional tasks. A user creates the desired sequence (Blowers' tree structure) by selecting graphical representations or icons from the tool boxes of FIG. 5. As the user is creating the sequence of functional tasks, the user can configure parameters of the functional tasks as illustrated in FIGS. 7 and 8 with respect to the blob and alignment vision tools, respectively. Once the desired sequence has been created, it can be stored or saved in a condensed method within an inspection sequence file 52 which is useable by the engine 46. The engine 46 takes the condensed stored sequence from the file 52 and executes it through the runtime screen of FIG. 9. (See Col. 8, line 61 – Col. 9, line 25)

Nowhere does Blowers teach or suggest automatically generating a graphical program comprising a plurality of interconnected nodes that visually indicate functionality of the graphical program, wherein the plurality of interconnected nodes are operable to perform the series of functional operations. In asserting that Blowers teaches this feature, the Examiner cites col. 8, lines 61-67, which reads:

> A design engine or task sequencer engine 46 is used to configure and test the flow and design of the application software as illustrated by an exemplary task sequencer list of FIG. 6. Graphical representations or icons are selected from the tool boxes of FIG. 5 which correspond to desired functional tasks and are linked into the tree structure of FIG. 6 by a task sequencer interface 50 in the desired locations.

Appellant notes that, as argued above, the task sequencer list of Figure 6, i.e., the tree structure of Figure 6, is not generated automatically, but rather via manual user-selection of icons from tool boxes. Nowhere does Blowers teach or suggest generating such a tree structure (or a graphical program) automatically. Thus, Blowers fails to teach this claimed feature.

Claim 49, 52

In addition to the distinctions noted above in regard to claim 48, Blowers fails to teach "wherein said receiving user input specifying the prototype is performed by a development environment; wherein the graphical program comprises second program instructions; wherein execution of the second program instructions is independent of execution of the development environment", as recited in claim 49.

In asserting that Blowers teaches this feature, the Examiner cites col. 9, lines 1-25, and col. 7, lines 35-39. As argued at length above, Blowers tree structure is not executable independently from the design engine 46, nor is the condensed sequence file (generated from the tree structure) executable independently from the design engine 46, and so Blowers does not, and cannot, teach this claimed feature, and so for at least these reasons, claim 49 is allowable.

Claim 56

In addition to the distinctions noted above in regard to claim 1, Blowers fails to teach "wherein said automatically generating the program comprises automatically generating source code to perform the functional operations of the prototype; wherein the source code is automatically generated in a format that allows a user to edit the automatically generated source code.", as recited in claim 56.

In asserting that Blowers teaches this feature, the Examiner cites col. 4, lines 64-67, which reads:

> The system creates jobs that are programmed through selecting and applying a sequence of tasks. The user is given the ability to access and change individual inspection task parameters.

As may be seen, the cited text discloses the user being allowed to edit task parameters, *not* automatically generated source code. Thus, Blowers fails to teach this claimed feature, and so claim 56 is allowable.

Section 103(a) Rejection

Claims 6, 19, and 32 stand rejected under 35 U.S.C. § 102(e) as being unpatentable over U.S. Patent No. 6,298,474 B1 to Blowers et al (henceforth, "Blowers"). Appellant

respectfully traverses this rejection for the following reasons. Different groups of claims are addressed under their respective subheadings.

Claims 6, 19, and 32

Applicant submits that in addition to the novel limitations of their respective base claims, claims 6, 19, and 32 recite further distinctions over the cited art. In asserting that Blowers teaches the features and limitations of these claims, the Examiner cites col. 3, lines 28-31, col. 3, lines 31-35, and col. 8, lines 61-67, which read as follows:

> "Then, the method includes receiving commands from a user to select desired hardware operating parameters corresponding to desired hardware and a machine vision graphical representation and its associated first control program corresponding to a desired machine vision task. The method includes displaying the structure. The selected machine vision graphical representation is a node of the structure. "
> ...
> A design engine or task sequencer engine 46 is used to configure and test the flow and design of the application software as illustrated by an exemplary task sequencer list of FIG. 6. Graphical representations or icons are selected from the tool boxes of FIG. 5 which correspond to desired functional tasks and are linked into the tree structure of FIG. 6 by a task sequencer interface 50 in the desired locations.

Thus, Blowers teaches receiving commands from a user to select desired parameters, but teaches nothing at all about receiving user input specifying which parameters are desired to have *associated graphical user interface elements*.

Applicant further submits that no proper motivation to modify Blowers has been provided. The only motivation suggest by the Examiner is simply a statement of the benefits and usefulness of this feature of Applicant's invention, which is not a proper motivation, and so may not be used make a prima facie case of obviousness.

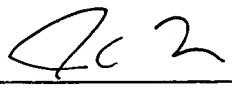
Moreover, Appellant respectfully submits that the respective base claims for claims 6, 19, and 32, have been shown above to be patentably distinct and non-obvious over Blowers, and so these dependent claims are similarly patentably distinct and non-obvious for at least the reasons provided above, and are thus allowable.

VIII. CONCLUSION

For the foregoing reasons, it is submitted that the Examiner's rejection of claims 1, 3-14, 16-27, 29-40, and 42-59 was erroneous, and reversal of the decision is respectfully requested.

The Commissioner is authorized to charge the appeal brief fee of $500.00 and any other fees that may be due to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5150-43700/JCH. This Appeal Brief is submitted with a return receipt postcard.

Respectfully submitted,



Jeffrey C. Hood
Reg. No. 35,198
ATTORNEY FOR APPLICANT(S)

Meyertons Hood Kivlin Kowert & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8800

Date: 5/5/2006 JCH/MSW

IX. CLAIMS APPENDIX

The following lists claims 1-59, incorporating entered amendments, as on appeal.

1. (Previously Presented) A method for generating a computer program, the method comprising:

receiving user input to a prototyping application, wherein the user input specifies a prototype, wherein the prototype comprises a series of functional operations, wherein the functional operations include a first functional operation with one or more associated parameters;

automatically generating a program that implements the prototype, in response to the specified prototype, wherein the program is operable to execute independently of the prototyping application;

wherein said automatically generating the program comprises automatically generating a graphical user interface for the program;

wherein said automatically generating the graphical user interface comprises automatically creating one or more graphical user interface elements associated with the one or more parameters of the first functional operation, wherein during execution of the program, at least one of the one or more graphical user interface elements is displayed and is operable to receive user input independently of the prototyping application.

2. (Cancelled)

3. (Previously Presented) The method of claim 1,

wherein said automatically generating the program comprises automatically generating source code for the program without direct user input specifying the source code.

4. (Previously Presented) The method of claim 1,

wherein the first functional operation has an associated input parameter;

wherein the one or more automatically created graphical user interface elements include a graphical user interface element for interactively receiving user input specifying a value for the input parameter.

5. (Previously Presented) The method of claim 1,

wherein the first functional operation has an associated output parameter;

wherein the one or more automatically created graphical user interface elements include a graphical user interface element for viewing program output indicating a value for the output parameter.

6. (Previously Presented) The method of claim 1, wherein the one or more parameters associated with the first functional operation comprises a plurality of parameters, the method further comprising:

receiving user input to the prototyping application specifying which of the plurality of parameters are desired to have associated graphical user interface elements;

wherein said automatically generating the graphical user interface comprises automatically creating a graphical user interface element associated with each specified parameter of the first functional operation, but not creating graphical user interface elements associated with unspecified parameters of the first functional operation.

7. (Previously Presented) The method of claim 1,

wherein the automatically generated program comprises a text-based program;

wherein automatically generating the program comprises automatically generating source code for the program in a text-based programming language.

8. (Previously Presented) The method of claim 1,

wherein the automatically generated program comprises a graphical program;

wherein automatically generating the program comprises automatically generating graphical source code for the graphical program, wherein the graphical source code includes a plurality of interconnected nodes that visually indicate functionality of the graphical program.

9. (Previously Presented) The method of claim 1,

wherein the prototyping application interfaces with a programming environment application in order to perform said generating the program.

10. (Previously Presented) The method of claim 1, wherein the first functional operation has a first parameter that has an associated data type, the method further comprising:

determining the data type of the first parameter;

wherein automatically creating the one or more graphical user interface elements comprises automatically creating a first graphical user interface element associated with the first parameter, wherein the first graphical user interface element is operable to receive user input according to the data type of the first parameter.

11. (Previously Presented) The method of claim 1,

wherein the prototype specifies an image processing algorithm;

wherein the automatically generated program implements the image processing algorithm.

12. (Previously Presented) The method of claim 11,

wherein the one or more automatically created graphical user interface elements include a first graphical user interface element for receiving user input specifying a value for an input parameter of the first functional operation, wherein the value for the input parameter affects the image processing algorithm.

13. (Previously Presented) The method of claim 11,

wherein the one or more automatically created graphical user interface elements include a first graphical user interface element for viewing an output value for an output parameter of the first functional operation, wherein the output value is determined by the image processing algorithm.

14. (Previously Presented) A system for generating a computer program, the system comprising:

a prototyping environment application for receiving user input specifying a prototype, wherein the prototype comprises a series of functional operations, wherein at least one of the operations has an associated one or more parameters;

wherein the prototyping environment application is operable to automatically generate a program that implements the prototype, in response to the specified prototype, wherein the program is operable to execute independently of the prototyping environment application;

wherein said automatically generating the program comprises automatically generating a graphical user interface for the program;

wherein said automatically generating the graphical user interface comprises automatically creating one or more graphical user interface elements associated with the one or more parameters, wherein during execution of the program, at least one of the one or more graphical user interface elements is displayed and is operable to receive user input independently of the prototyping environment application.

15. (Cancelled)

16. (Previously Presented) The system of claim 14,

wherein said automatically generating the program comprises automatically generating source code for the program without direct user input specifying the source code.

17. (Previously Presented) The system of claim 14,

wherein at least one of the operations has an associated input parameter;

wherein the one or more automatically created graphical user interface elements include a graphical user interface element for interactively receiving user input specifying a value for the input parameter.

18. (Previously Presented) The system of claim 14,

wherein at least one of the operations has an associated output parameter;

wherein the one or more automatically created graphical user interface elements include a graphical user interface element for viewing program output indicating a value for the output parameter.

19. (Previously Presented) The system of claim 14,

wherein a plurality of parameters are associated with the functional operations;

wherein the prototyping environment application is operable to receive user input specifying which of the plurality of parameters are desired to have associated graphical user interface elements;

wherein said generating the graphical user interface comprises creating a graphical user interface element associated with each specified parameter, but not creating graphical user interface elements associated with unspecified parameters.

20. (Previously Presented) The system of claim 14,

wherein the automatically generated program comprises a text-based program;

wherein automatically generating the program comprises automatically generating source code for the program in a text-based programming language.

21. (Previously Presented) The system of claim 14,

wherein the automatically generated program comprises a graphical program;

wherein automatically generating the program comprises automatically generating graphical source code for the graphical program, wherein the graphical source code includes a plurality of interconnected nodes that visually indicate functionality of the graphical program.

22. (Original) The system of claim 14,

wherein the prototyping environment application interfaces with a programming environment application in order to perform said generating the program.

23. (Previously Presented) The system of claim 14,

wherein at least one parameter has an associated data type;

wherein the prototyping environment application is operable to determine the data type of the at least one parameter;

wherein creating a graphical user interface element associated with the at least one parameter comprises creating a graphical user interface element according to the data type of the at least one parameter.

24. (Previously Presented) The system of claim 14,

wherein the prototyping environment application is an image processing prototype environment application;

wherein the prototype specifies an image processing algorithm;

wherein the automatically generated program implements the image processing algorithm.

25. (Previously Presented) The system of claim 24,

wherein the one or more automatically created graphical user interface elements include one or more graphical user interface elements for receiving input parameter values affecting the image processing algorithm.

26. (Previously Presented) The system of claim 24,

wherein the one or more automatically created graphical user interface elements include one or more graphical user interface elements for viewing output parameter values determined by the image processing algorithm.

27. (Previously Presented) A memory medium comprising program instructions executable to:

receive user input to a prototyping application, wherein the user input specifies a prototype, wherein the prototype comprises a series of functional operations, wherein at least one of the operations has an associated one or more parameters;

automatically generate a program that implements the prototype, in response to the specified prototype, wherein the program is operable to execute independently of the prototyping application;

wherein said automatically generating the program comprises automatically generating a graphical user interface for the program;

wherein said automatically generating the graphical user interface comprises automatically creating one or more graphical user interface elements associated with the one or more parameters, wherein during execution of the program, at least one of the one or more graphical user interface elements is displayed and is operable to receive user input independently of the prototyping application.

28. (Cancelled)

29. (Previously Presented) The memory medium of claim 27,

wherein said automatically generating the program comprises automatically generating source code for the program without direct user input specifying the source code.

30. (Previously Presented) The memory medium of claim 27,

wherein at least one of the operations has an associated input parameter;

wherein the one or more automatically created graphical user interface elements include a graphical user interface element for interactively receiving user input specifying a value for the input parameter.

31. (Previously Presented) The memory medium of claim 27,

wherein at least one of the operations has an associated output parameter;

wherein the one or more automatically created graphical user interface elements includea graphical user interface element for viewing program output indicating a value for the output parameter.

32. (Previously Presented) The memory medium of claim 27, wherein a plurality of parameters are associated with the functional operations, wherein the program instructions are further executable to:

receive user input to the prototyping application specifying which of the plurality of parameters are desired to have associated graphical user interface elements;

wherein said generating the graphical user interface comprises creating a graphical user interface element associated with each specified parameter, but not creating graphical user interface elements associated with unspecified parameters.

33. (Previously Presented) The memory medium of claim 27,

wherein the automatically generated program comprises a text-based program;

wherein automatically generating the program comprises automatically generating source code for the program in a text-based programming language.

34. (Previously Presented) The memory medium of claim 27,

wherein the automatically generated program comprises a graphical program;

wherein automatically generating the program comprises automatically generating graphical source code for the graphical program, wherein the graphical source code includes a plurality of interconnected nodes that visually indicate functionality of the graphical program.

35. (Previously Presented) The memory medium of claim 27,

wherein the prototyping application interfaces with a programming environment application in order to perform said automatically generating the program.

36. (Previously Presented) The memory medium of claim 27, wherein at least one parameter has an associated data type, wherein the program instructions are further executable to determine the data type of the at least one parameter;

wherein creating a graphical user interface element associated with the at least one parameter comprises creating a graphical user interface element according to the data type of the at least one parameter.

37. (Previously Presented) The memory medium of claim 27,

wherein the prototype specifies an image processing algorithm;

wherein the automatically generated program implements the image processing algorithm.

38. (Previously Presented) The memory medium of claim 37,

wherein the one or more automatically created graphical user interface elements include one or more graphical user interface elements for receiving input parameter values affecting the image processing algorithm.

39. (Previously Presented) The memory medium of claim 37,

wherein the one or more automatically created graphical user interface elements include one or more graphical user interface elements for viewing output parameter values determined by the image processing algorithm.

40. (Previously Presented) A computer-implemented method for automatically generating a computer program, the method comprising:

receiving program information to a first application, wherein the program information specifies functionality of the computer program;

automatically generating the computer program in response to the program information, wherein the computer program implements the specified functionality, wherein the program is operable to execute independently of the first application;

wherein said automatically generating the program comprises automatically generating a graphical user interface for the program;

wherein said automatically generating the graphical user interface comprises automatically creating one or more graphical user interface elements for providing input to and/or viewing output from the program, wherein during execution of the program, the one or more graphical user interface elements are displayed and at least one of the one or more graphical user interface elements is operable to receive user input independently of the first application.

41. (Cancelled)

42. (Previously Presented) The method of claim 40,

wherein said automatically generating the computer program comprises automatically generating source code for the program without direct user input specifying the source code.

43. (Previously Presented) The method of claim 40,

wherein each of the one or more automatically created graphical user interface elements corresponds to one or more parameters specified by the program information.

44. (Previously Presented) The method of claim 40,

wherein the generated computer program comprises a graphical program, wherein automatically generating the computer program comprises automatically generating a plurality of interconnected nodes that visually indicate functionality of the graphical program.

45. (Original) The method of claim 40,

wherein the received program information specifies one of:

a prototype;

a test executive sequence; and

a state diagram.

46. (Previously Presented) The method of claim 1, wherein said automatically generating the program comprises:

automatically generating a block diagram, wherein the block diagram comprises a plurality of interconnected nodes that visually indicate functionality of the program.

47. (Previously Presented) The method of claim 1, wherein said generating the graphical user interface comprises:

automatically generating a user interface panel, wherein the user interface panel comprises the one or more graphical user interface elements.

48. (Previously Presented) A method for generating a computer program, the method comprising:

receiving user input specifying a prototype, wherein the prototype comprises a series of functional operations, wherein at least one of the operations has an associated one or more parameters;

in response to said receiving user input specifying the prototype, automatically generating a graphical program, wherein automatically generating the graphical program comprises automatically generating a plurality of interconnected nodes that visually indicate functionality of the graphical program, wherein the plurality of interconnected nodes are operable to perform the series of functional operations;

wherein said automatically generating the graphical program comprises automatically generating a graphical user interface for the graphical program, wherein the graphical user interface for the graphical program comprises at least one graphical user interface element which is associated with at least one of the one or more parameters;

wherein the graphical program is interpretable or compilable.

49. (Previously Presented) The method of claim 48,

wherein said receiving user input specifying the prototype is performed by a development environment;

wherein the graphical program comprises second program instructions;

wherein execution of the second program instructions is independent of execution of the development environment.

50. (Previously Presented) A method for generating a computer program, the method comprising:

receiving user input specifying a prototype, wherein the prototype comprises a series of functional operations, wherein at least one of the operations has an associated one or more parameters;

in response to said receiving user input specifying the prototype, automatically generating a graphical program, wherein automatically generating the graphical program comprises automatically generating a plurality of interconnected nodes that visually indicate functionality of the graphical program, wherein said automatically generating the graphical program further comprises automatically generating a graphical user interface for the graphical program; and

associating at least one of the one or more parameters with an element of the graphical user interface.

51. (Previously Presented) The method of claim 50, further comprising:

receiving user input indicating the at least one of the one or more parameters;

wherein said associating the at least one of the one or more parameters with the element of the graphical user interface is based on said receiving user input indicating the at least one of the one or more parameters.

52. (Previously Presented) The method of claim 51,

wherein said receiving user input specifying the prototype is performed by first program instructions;

wherein the graphical program comprises second program instructions;

wherein execution of the second program instructions is independent of execution of the first program instructions.

53. (Previously Presented) A method for generating a computer program, the method comprising:

displaying a prototyping environment user interface on a display of a computer system, wherein the prototyping environment user interface is usable to create a prototype;

receiving user input specifying the prototype, wherein the prototype comprises a series of functional operations, wherein at least one of the operations has an associated one or more parameters; and

automatically generating a program that implements the prototype, in response to the specified prototype;

wherein said automatically generating the program comprises automatically generating a graphical user interface for the program;

wherein the graphical user interface of the program comprises at least one graphical user interface element which is associated with at least one of the associated one or more parameters, wherein the at least one graphical user interface element performs at least one of receiving information to the program and outputting information from the program during execution of the program, wherein the graphical user interface of the program is independent of the prototyping environment user interface.

54. (Previously Presented) The method of claim 53, wherein the program is interpretable or compilable.

55. (Previously Presented) A method for generating a computer program, the method comprising:

receiving user input to a development environment, wherein the user input specifies a series of functional operations, wherein at least one of the operations has an associated one or more parameters; and

automatically generating a program that implements the series of functional operations, in response to the user input, wherein program execution of the program is independent of execution of the development environment;

wherein said automatically generating the program comprises automatically generating a graphical user interface for the program;

wherein the graphical user interface of the program comprises at least one graphical user interface element which is associated with at least one of the associated one or more parameters, wherein the at least one graphical user interface element performs one or more of receiving information through the graphical user interface and outputting information from through the graphical user interface during execution of the program;

wherein the program is interpretable or compilable.

56. (Previously Presented) The method of claim 1,

wherein said automatically generating the program comprises automatically generating source code to perform the functional operations of the prototype;

wherein the source code is automatically generated in a format that allows a user to edit the automatically generated source code.

57. (Previously Presented) The method of claim 1,

wherein the functional operations also include a second functional operation with one or more associated parameters;

wherein said automatically generating the graphical user interface further comprises automatically creating one or more graphical user interface elements associated with the one or more parameters of the second functional operation.

58. (Previously Presented) The method of claim 1, further comprising:

executing the automatically generated program, wherein said executing the automatically generated program comprises displaying the automatically generated graphical user interface independently of the prototyping application.

59. (Previously Presented) The method of claim 1,

wherein the automatically generated program is operable to perform the functional operations of the prototype.

X. EVIDENCE APPENDIX

No evidence submitted under 37 CFR §§ 1.130, 1.131 or 1.132 or otherwise entered by the Examiner is relied upon in this appeal.

XI. RELATED PROCEEDINGS APPENDIX

There are no related proceedings.